Gap heuristics and tree construction using gaps

Gap Heuristics and Tree Construction using gaps

Chantal Korostensky and Gaston H. Gonnet
Institute for Scientific Computing
ETH Zurich, Switzerland
e-mail: {gonnet,korosten}@inf.ethz.ch

Abstract

The construction of multiple sequence alignments (MSAs) is a fundamental problem in biology. Yet the problem cannot be solved exactly in most cases, as it is exponential in the number of sequences. It is therefore highly desirable to develop heuristics. It is reasonable that not every heuristic starts from scratch, but uses the expertise that has evolved over the years. We present a set of heuristics that take as input an MSA that was produced from any algorithm and produces an improved MSA. The scoring of the resulting alignment is based on a probabilistic model that we developed [16]. With this scoring function we can determine the upper bound (maximum possible score), so we know when the MSA is optimal.

The heuristics use the fact that if the sequences in the MSA are related, there exists an (unknown) evolutionary tree. It treats gaps as a special character as gaps play a special role in both protein structure and evolutionary history in order to improve the alignment. From an optimized MSA gaps can be used to reconstruct evolutionary trees.

1 Introduction

Several structure prediction methods are based on multiple sequence alignments (MSAs) [10, 11, 21, 9, 2, 22]. The quality of an MSA highly influences the outcome of structure prediction. As a first step the MSA is scanned for parse regions. Parses are regions in the protein between secondary structures ($\alpha-$helices or $\beta-$sheets), such as loops. Usually gaps (insertions or deletion events) indicate such parses. This is the most important step, and misplaced gaps are a disaster for structure prediction, because once a parse region is assigned, no other secondary structures can be predicted there anymore.

The problem of calculating MSAs can be viewed as the problem of inserting gaps at the correct places. There are many MSA algorithms [26, 19, 12, 18, 14], but most of them do not take into account that gaps play a special role both in terms of structure and evolution (see following sections). It is important to develop heuristics that use as much biological information as possible, such that structure prediction based on MSA becomes more reliable.

```
A: RPCVCP___VLRQAAQ__QVLQRQIIQGPQQLRRLF_AA
B: RPCACP___VLRQVVQ__QALQRQIIQGPQQLRRLF_AA
C: KPCLCPKQAAVKQAAH__QQLYQGQLQGPKQVRRAFRLL
D: KPCVCPRQLVLRQAAHLAQQLYQGQ____RQVRRAF_VA
E: KPCVCPRQLVLRQAAH__QQLYQGQ____RQVRRLF_AA
```

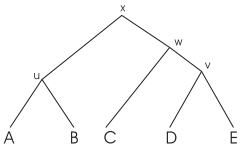

Figure 1: The sequences $\{A, B, C, D, E\}$ of the MSA on the left are related by the evolutionary tree on the right.

Example 1.1: In the MSA in Figure 1 five sequences are aligned ($n = 5$). We assume that the sequences are related and that we have the correct tree. The internal nodes in the tree represent unknown ancestor sequences.

Definition 1.1 *Given is a set of sequences $S = \{s_1, .., s_n\}$ with $s_i \in \Sigma^*$ where Σ is a finite alphabet. A Multiple Sequence Alignment (MSA) consists of a set of sequences $\mathrm{A} = \langle a_1, a_2, .., a_n \rangle$ with $a_i \in \Sigma'^*$ where $\Sigma' = \Sigma \cup \{"\_"\} \notin \Sigma$. $\forall a_i \in \mathrm{A} : |a_i| = k$. The sequence obtained from $a_i \in \mathrm{A}$ by removing all "_" gap characters is equal to s_i.*

Definition 1.2 *The character "_" or any contiguous sequence of "_" within an aligned sequence $a_i \in \mathrm{A}$ is called a gap. A gap corresponds to an insertion or deletion event (indel). The position of a gap g in sequence a_i is $pos(g)$, the length of a gap $len(g)$ is the number of dashes.*

Definition 1.3 *The tree $T(S) = (V, E, L)$ is a binary, leaf labeled tree with leafset $L = \{s_1, .., s_n\}$.*

In our context a tree $T(S)$ associated with a set of sequences $S = \{s_1, .., s_n\}$ is the tree that corresponds to the evolutionary history of the sequences of S. The internal nodes V represent (usually unknown) ancestor sequences.

Definition 1.4 *An MSA scoring function is a function* $F : \mathrm{A} \rightarrow I\!R$.

Definition 1.5 *Let* $\mathcal{A}$ *be the set of all possible MSAs that can be generated for a given set of sequences* $S = \{s_1, s_2, .., s_n\}$. *The optimal MSA* $\bar{\mathrm{A}} \in \mathcal{A}$ *is an MSA such that w.l.o.g.* $F(\bar{\mathrm{A}}) = \max_{\mathrm{A} \in \mathcal{A}} F(\mathrm{A})$. *In many scoring functions the minimum is the optimal.*

Problem 1.1 (MSA problem) *Given is a set of sequences* $S = \{s_1, .., s_n\}$. *Find the optimal MSA* A *for* S.

Both, the evolutionary history of the proteins considered and the structure of the proteins are correlated with the MSA. We use those two aspects to explain the ideas of the heuristics. We then present a set of heuristics that are derived from those biological observations to improve MSAs and show how to reconstruct an evolutionary tree from the gaps.

1.1 Scoring functions

1.1.1 Scoring of Pairwise Sequence Alignments

Definition 1.6 *The optimal pairwise alignment* $OPA(s_1, s_2)$ *of two sequences* s_1, s_2 *is an alignment with the maximum score where a probabilistic scoring method [5, 15] is used. We refer to a pairwise alignment of two sequences* s_1, s_2 *with* $\langle s_1, s_2 \rangle$.

Usually the optimal score is determined via standard dynamic programming [24, 17]. An affine gap cost is used according to the formula $a + l \cdot b$, where a is a fixed gap cost, l is the length of the gap and b is the incremental cost [3]. Note that scores represent the probabilities that the two sequences have a common ancestor. The larger the score is the more likely it is that the two sequences are homologous and therefore have a common ancestor.

1.1.2 Scoring of MSAs

Our scoring function $F(\mathrm{A})$ of an MSA is similar to the sum-of-pairs measure [4], except that we do not add all scores of all pairwise alignments, but only the scores of the pairwise alignments in the TSP order $C(S)$ divided by two. A TSP order is a circular order with respect to the associated evolutionary tree [16].

Problem 1.2 (TSP problem) *Given is a set of sequences* $S = \{s_1, .., s_n\}$ *and the corresponding scores of the optimal pairwise alignments. The problem is to find the longest tour where each sequence is visited once.*

Definition 1.7 *The TSP order* $C(S)$ *of set of sequences* $S = \{s_1, .., s_n\}$ *is the order of the sequences that is derived from the optimal solution of a TSP.*

Definition 1.8 *The function* $S(x, y)$ *scores two symbols* $x, y \in \Sigma'$:

$$S(x,y) = \begin{cases} DM(x,y), \text{ if } x \neq \text{"\_"} and y \neq \text{"\_"} \\ 0 \text{ if } x = \text{"\_"} and y = \text{"\_"} \\ \text{otherwise an affine gap cost that depends on the gap length [3]} \end{cases}$$

DM is an entry in a Dayhoff matrix [15].

The function $S(x, y)$ (see Definition 1.8) scores two symbols x, y that are either amino acids or gap characters in our case.

Definition 1.9 *The score of the induced MSA-derived pairwise alignment* $\underline{MPA(a_i, a_i)}$ *of two sequences* $a_i, a_j \in \mathrm{A}$ *is:*

$$MPA(a_1, a_2) = \sum_{m=1}^{k} S(a_i[m], a_j[m])$$

The CS score $CS(\mathrm{A})$ of an MSA A is the sum of MPA scores in the TSP order C divided by two:

Definition 1.10 *The score* $\underline{CS(\mathrm{A})}$ *of an MSA* A *is defined as:* $CS(\mathrm{A}) = \frac{1}{2} \sum_{i=1}^{n} MPA(a_{C_i}, a_{C_{i+1}})$ *where* $C_{n+1} = C_1$.

Example 1.2: The table to the right in Figure 2 shows the pairwise MPA scores of the sequences from the MSA on the left. It is easy to verify that the longest tour is (A, B, C, D, E, A) and that it is a circular tour in the corresponding tree (see Figure 1). The score of the MSA is the sum of pairwise alignments in circular order divided by two, so $F(\mathrm{A}) = (336 + 79 + 171 + 327 + 110)/2 = 512$. Here the MSA has the optimal score.

```
A: RPCVCP___VLRQAAQ__QVLQRQIIQGPQQLRRLF_AA
B: RPCACP___VLRQVVQ__QALQRQIIQGPQQLRRLF_AA
C: KPCLCPKQAAVKQAAH__QQLYQGQLQGPKQVRRAFRLL
D: KPCVCPRQLVLRQAAHLAQQLYQGQ____RQVRRAF_VA
E: KPCVCPRQLVLRQAAH__QQLYQGQ____RQVRRLF_AA
```

	A	B	C	D	E
A	0	336	27	44	110
B	336	0	79	56	99
C	27	79	0	171	176
D	44	56	171	0	327
E	110	99	176	327	0

Figure 2: An MSA and a table containing the pairwise scores

Note that if there is a deletion in both sequences, there is no penalty (score is zero) because that deletion happened in some ancestor, and its penalty has been counted already.

2 Biological background

Gaps play a special role both in the evolutionary history, in the problem of tree construction and in the calculation of MSAs. Whereas mutations happen very frequently and even back mutations happen often when sequences are far apart, this is not true for insertions and deletions (indels). Indels appear less frequently and have a much greater impact on the resulting protein than a mutation. Hence back mutations are hardly observed at all. This observation is very useful if sequences are compared that are very distant, when there are so many mutations that it becomes very hard to see their relationship. In the Markovian model of evolution, we assume that mutations can happen anywhere in a protein and are independent of each other. This is not entirely true, but as an approximation it is a useful and logical assumption, as one mutation usually does not have such a great impact on the protein structure. This is not true for gaps, hence it makes sense to treat them separately and use as much information as we can from biology to develop better heuristics.

2.1 Gaps and protein structure

Indels appear more frequently in loops than in alpha helices or beta strands, as they do not have a great impact on the protein structure when they are in loops. Statistically, when looking at a set of homologous (related) proteins, indels in loops usually appear around a "center", i.e. when a loop is shortened, this usually does not happen at the end but around the middle of the loop, and vice versa if a loop is deleted or partially deleted, this also happens around the center of the loop (see figure 3).

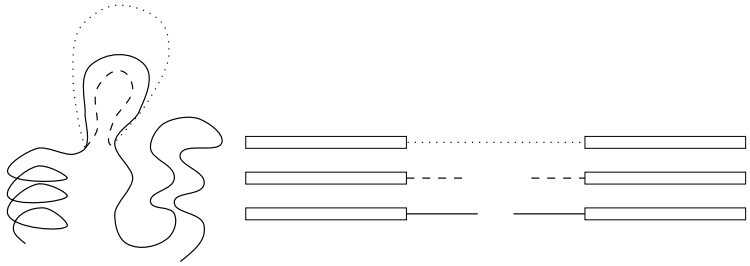

Figure 3: gaps in turns

Since we usually don't know the structure of a protein, we cannot know where the loops are. But when we build an MSA and find several gaps of similar sizes at reasonable distance from each other (reasonable meaning not more than the average domain size, i.e. 30 to 50 amino acids), we can *assume* that the indels are in a loop. This is how MSAs are scanned for secondary structure prediction and those regions are often called *parses*. So one idea of a heuristic is to use this knowledge and push the gaps together. Of course, the final judgement is always done by scoring the resulting MSA. The changes are kept only of the score (and therefore the probability, see Definition 1.10) of the MSA improves.

Definition 2.1 *A gap block B is a set of gaps $\{g_1 \in a_1, .., g_k \in a_k\}$ where the gaps have the following properties: $\forall g_i, g_j \in B : pos(g_i) = pos(g_j)$ and $len(g_i) = len(g_j)$.*

We will use the term gap block later and explain it in more detail.

Example 2.1: The following alignment was produced with the probabilistic model [14]. There are several gaps close to each other, and if we look closely at the alignment (or ask an expert), we can see that the alignment can be improved by shifting the gap blocks together. To score alignments we use the CS score (see Definition 1.10) derived in [16]. A larger score refers to a better alignment. The maximum possible score is the score derived from the optimal pairwise alignments (see Definition **??**).

```
Score of the alignment: 2325.268
Maximum possible score: 2472.632

16      PDCVCPTLKQAAKAVRL___________QGQHQPMQVRKIYQTAKHLPNVC
9       PVCVCPTLRQAARAVSL___________QGQHGPFQSRKIYKTAKYLPNIC
8       PVCVCPTLKQAARAVSL___________QGQHGPFQSRKIYQSAKYLPNIC
13      QVCVCPTLKQAAKSVRV___________QGQHGPFQSTRIYQIAKNLPNVC
10      PLCVCPTLKGASKAVKQQVRQQLEQQGQQ___GPHVISRIYQTATHLPKVC
```

```
7       PLCVCPTLKGASKAVKQQVRQQQGQQGQQ___LQQVISRIYQTATHLPKVC
3       PLCVCPTLKGASKAVRQQVRQQQ_GQQMQGQQMQQVISRVYQTATHLPRVC
6       PLCVCPTLKGASKAVKQQIRQQGQQQQGQQGQQLQHEISRIYQTATHLPRVC
4       PLCVCPTLKGAAKAVKQQIQQQGQQHGQQGQQLQHEIRRIYQTATHLPKVC
12      PLCVCPTLKGASKAVKQQIQQQGQQQGKLQMV_____SRIYQTATHLPKVC
5       PLCVCPTLKGASKAVKQQIQQQGQQQGKQQMV_____SRIYQTATHLPKVC
11      PLCVCPTLRGASKAVKQQIQQQEQQQGKQQMV_____NRIYQTATHLPKVC
```

The alignment below with the stacked gaps looks better, has a better score and also makes more sense from a structural point of view:

Example 2.2:

```
Score of the alignment: 2389.013
Maximum possible score: 2472.632

16      PDCVCPTLKQAAKAVRLQ__________GQHQPMQVRKIYQTAKHLPNVC
9       PVCVCPTLRQAARAVSLQ__________GQHGPFQSRKIYKTAKYLPNIC
8       PVCVCPTLKQAARAVSLQ__________GQHGPFQSRKIYQSAKYLPNIC
13      QVCVCPTLKQAAKSVRVQ__________GQHGPFQSTRIYQIAKNLPNVC
10      PLCVCPTLKGASKAVKQQVRQQLE___QQGQQGPHVISRIYQTATHLPKVC
7       PLCVCPTLKGASKAVKQQVRQQQG___QQGQQLQQVISRIYQTATHLPKVC
3       PLCVCPTLKGASKAVRQQVRQQQG_QQMQGQQMQQVISRVYQTATHLPRVC
6       PLCVCPTLKGASKAVKQQIRQQGQQQGQQGQQLQHEISRIYQTATHLPRVC
4       PLCVCPTLKGAAKAVKQQIQQQGQQHGQQGQQLQHEIRRIYQTATHLPKVC
12      PLCVCPTLKGASKAVKQQIQQQG_____QQQGKLQMVSRIYQTATHLPKVC
5       PLCVCPTLKGASKAVKQQIQQQG_____QQQGKQQMVSRIYQTATHLPKVC
11      PLCVCPTLRGASKAVKQQIQQQE_____QQQGKQQMVNRIYQTATHLPKVC
```

2.2 Gaps and Evolution

2.2.1 Gaps and Trees

Assume we are given an optimal MSA A and assume further that we know the correct tree $T(\mathrm{A})$. Insertions and deletions (indels) are evolutionary events that can be placed in the evolutionary tree (see Figure 4). The result of a deletion event are gaps that appear in all sequences $a_i \in \mathrm{A}$ of the subtree where the event took place. In contrast, when the event was an insertion, gaps appear in all sequences that are not in the subtree below the event.

Example 2.3: In the alignment of Figure 4, there are 4 indel events and therefore 4 gap blocks. The first gap block consists of two gaps that appear in sequences A and B. We assume we have the correct tree and we also assume that we know the place of the root. In this case, the event must be a deletion in the ancestor of sequences A and B. Accordingly, event 2 is an insertion in sequence D, event 3 represents a deletion in the ancestor of D and E, and finally event 4 is an insertion in sequence E.

Note that the sequences of Figure 4 are ordered in a circular way $C(T)$. Whenever sequences are in a circular order with respect to the evolutionary tree, the gaps that

```
Event     1         2         3         4
A: RPCVCP___VLRQAAQ__QVLQRQIIQGPQQLRRLF_AA
B: RPCACP___VLRQVVQ__QALQRQIIQGPQQLRRLF_AA
C: KPCLCPKQAAVKQAAH__QQLYQGQLQGPKQVRRAFRLL
D: KPCVCPRQLVLRQAAHLAQQLYQGQ____RQVRRAF_VA
E: KPCVCPRQLVLRQAAH__QQLYQGQ____RQVRRLF_AA
```

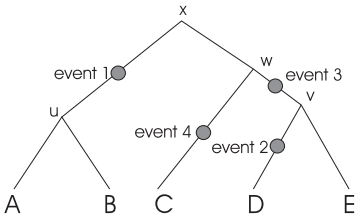

Figure 4: Insertion and deletion events are shown in the evolutionary tree (right) of the MSA (left).

belong to the same evolutionary event appear in a gap block (ignoring the breaking of the circularity).

2.3 Classification of the problems

Usually MSA algorithms do not produce the optimal alignment and therefore gaps are often misplaced. When gaps or gap blocks that would really belong to same evolutionary event are not recognized as such in the MSA, there are three types of misplacement of gaps:

- shifted gaps: gap blocks of same size appear shifted horizontally towards each other
- split gaps: gap blocks are cut apart vertically and appear as two gap blocks
- a combination of the above

2.3.1 Shifted gaps

Very often algorithms produce MSAs where gaps of the same size appear slightly shifted horizontally. From a biological point of view there are two possibilities:

a) the gaps are in a loop. Since gaps appear frequently in loops, it is possible (likely) that two events produce indels of the same size at the same place. So it makes sense to group the gaps for structural reasons.

b) the gaps are *not* in a loop. Since gaps do not appear so frequently outside of loops, it is not very likely that two $different$ events produce gaps of the same size at the same place. It is more likely that it was just one event and that the algorithm that produced the alignment did not find the correct alignment. So in this case it also makes sense to group the gaps together.

When gaps are grouped together, e.g gaps 1 and 1' from example 2.4, the number of evolutionary events is reduced. This can be viewed as "pushing" and fusing the events up in the tree, if the corresponding tree would be known (see figure 5). Each gap has a high *cost*, so reducing the number of indel events reduces the cost and therefore increases the score in most cases. In the maximum likelihood approach we are generally interested

in finding the most likely evolutionary configuration, that is the configuration with the maximum score.

Example 2.4: In this example gaps 1 and $1'$ have the same size (length 3) and appear at almost the same place. When the gaps are grouped together, both the score increases and the alignment looks better (if we ask an expert).

```
Score:   135.466                        Score:   240.139
Maximum: 240.139                        Maximum: 240.139

Event 1   1'                           Event     1
A: RP___PCVCVLRQ                        A: RPCVCP___VLRQ
B: RPCACP___VLRQ                        B: RPCACP___VLRQ
C: KPCVCPRQLVLQ                 =>      C: KPCVCPRQLVLRQ
D: KPCVCPRQLVLRQ                        D: KPCVCPRQLVLRQ
E: KPCLCPKQAAVKQ                        E: KPCLCPKQAAVKQ
```

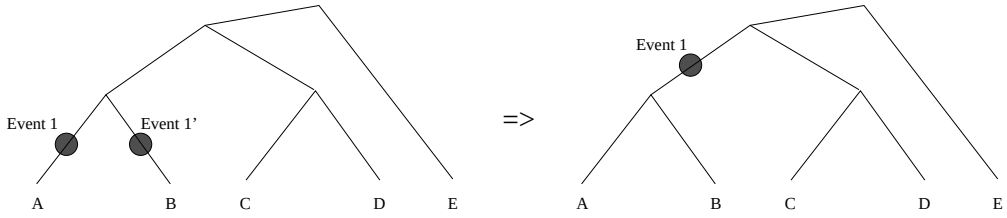

Figure 5: When gaps of same size but from different sequences are grouped together, this corresponds to "pushing" the event up in the evolutionary tree

2.3.2 Split gaps

Sometimes a gap that is caused by just one indel event is split into several gaps in the MSA. The problem is now to recognize such a situation. Assume there is a sequence of gaps $b_i = \{g_1, .., g_k\}$ in an aligned sequence a_i, and we want to verify if this is a gap that is split apart or if the gaps really are different indel events. One possibility to verify this is simply by fusing the gaps and to score the alignment again. If the score increases, we keep the change. But since we do not want to try out all possibilities, as the number of possible ways to fuse and shift gaps would increase exponentially by the number of gaps, there is a simple algorithm:

Definition 2.2 *The function gapsum(b) for a set of gaps* $b = \{g_1, .., g_k\}$ *is the sum of all dashes of all* $g \in b$.

a) get the $gapsum(b_i)$ for the set of gaps $b_i \in a_i$.

b) get other sequences of gaps b_j of a_j, where the gaps $g \in b_j$ are *close* to the gaps $g' \in b_i$. *Close* means that there is a parameter e such that $\forall g \in b_i, \forall g' \in b_j$: $|pos(g') - pos(g)| \leq e$.

c) If there are other sequences of gaps b_j where $gapsum(b_i) = gapsum(b_j)$, it is likely that those events belong to just one event and the gaps should be grouped.

When two gaps g_i and g_j are depicted in a tree that actually belong to the same indel event, they appear on the same edge in the tree. When the gaps are combined then only one event is left (see figure 6).

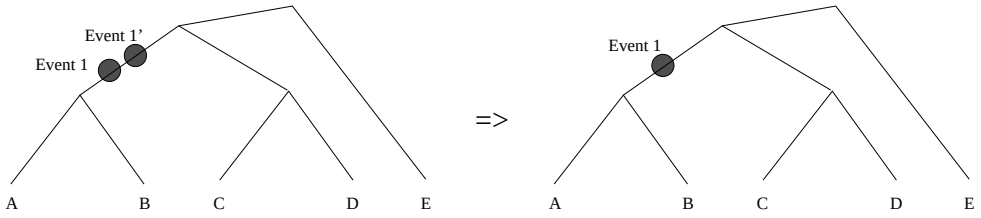

Figure 6: Gap fusion

Example 2.5: In the MSA below, sequence A has two gaps, so $b_A = \{g_{A1}, g_{A2}\}$, and $gapsum(b_A) = 3$ (the total number of dashes). To figure out if this is just one event, we need to look at other sequences. Sequence B has 3 gaps ($b_B = \{g_{B1}, g_{B2}, g_{B3}\}$), and the $gapsum(b_B)$ is also 3. In addition, they appear close to the gaps in the first sequence. If we push the gaps together and move them to the same position, the resulting alignment has a better score (the optimal score in this case), a smaller number of events and also looks better from a biological point of view.

```
Score:   104.635                      Score:   240.139
Maximum: 240.139                      Maximum: 240.139

Event 1    1'                          Event     1
A: RPC__VCP_VLRQ                       A: RPCVCP___VLRQ
B: RPC_A_CPV_LRQ                       B: RPCACP___VLRQ
C: KPCVCPRQLVLRQ               =>      C: KPCVCPRQLVLRQ
D: KPCVCPRQLVLRQ                       D: KPCVCPRQLVLRQ
E: KPCLCPKQAAVKQ                       E: KPCLCPKQAAVKQ
```

3 Methods

Before we describe the algorithms we first want to give a short summary of the above sections:

a) Influence of indels in terms of protein structure:

1. If there are two gaps of the same size close to each other in an MSA A, they corresponding indel events happened most likely in a loop or coil in the protein, because indel events hardly ever appear in alpha helices or beta sheets. Hence it is reasonable to group those gaps together in the MSA (if the score increases).

2. Since gaps most likely appear in loops, even gaps of different sizes that are close to each other should be grouped around the center of the loop.

b) Influence of indels in terms of evolution:

1. Two indel events of the same size at the same place in a protein (or nearby) are very unlikely. So if there are two gaps of same size at almost the same place in an MSA A, it is more likely that the gaps belong to just one indel event and simply were not placed correctly by whatever algorithm was used. Therefore such gaps should be moved to the same place (if the score increases).
2. back mutations are very unlikely, that is, that a deletion is "canceled" by an insertion or vice versa. So i.e. if the construction of an evolutionary tree is done using maximum parsimony, then using gaps should be more reliable than using amino acids. Therefore it is reasonable to use gaps to reconstruct evolutionary trees, especially if the proteins are very distant.

3.1 Finding all gap blocks

Problem 3.1 (Finding gap blocks) *Given is an MSA* $\mathrm{A} = \{a_1, .., a_n\}$. *The problem is to find all possible gap blocks* $B \in \mathrm{A}$ *(see Definition 3.2).*

To find all possible gap blocks B we transform the MSA A to a list L of integers:

Each gap is represented by an integer that is the sum of the dashes $len(g)$. All amino acids are removed. This way each sequence $a_i \in \mathrm{A}$ is transformed to a list of integers $l \in I\!N^m$. The whole alignment is transformed into a list of l: $L = \{l_1, .., l_n\}$.

Example 3.1: The MSA in Figure 7 is transformed into a list L:

```
16      PDCVCPTLKQAAKAV____RLQG_QHQ____PMQ_______VRKIYQTAKHLPNVC
9       PVCVCPTLRQAARAV____SLQG_QHG____PFQ_______SRKIYKTAKYLPNIC
8       PVCVCPTLKQAARAV____SLQG_QHG____PFQ_______SRKIYQSAKYLPNIC
13      QVCVCPTLKQAAKSVRVQGQHG_________________PFQSTRIYQIAKNLPNVC
10      PLCVCPTLKGASKAVKQQVRQQLEQQG____Q____QGPHVISRIYQTATHLPKVC
7       PLCVCPTLKGASKAVKQQVRQ___QQGQQQGQQLQQV_____ISRIYQTATHLPKVC
3       PLCVCPTLKGASKAVRQQVRQ___QQG___QQMQGQQMQQVISRVYQTATHLPRVC
6       PLCVCPTLKGASKAVKQQIRQQGQQQGQQGQQLQHE_____ISRIYQTATHLPRVC
4       PLCVCPTLKGAAKAVKQQIQQQGQQHGQQGQQLQHE_____IRRIYQTATHLPKVC
12      PLCVCPTLKGASKAVKQQIQQQGQQQG__KLQM________VSRIYQTATHLPKVC
5       PLCVCPTLKGASKAVKQQIQQQGQQQG__KQQM________VSRIYQTATHLPKVC
11      PLCVCPTLRGASKAVKQQIQQQEQ__QQGKQQM________VNRIYQTATHLPKVC
```

```
4 1 4 8
4 1 4 8
4 1 4 8
16
4 4
3 5
3 3
5
5
2 8
2 8
2 8
```

Figure 7: Each sequence of the MSA is transformed to a list l of integers

We need to redefine gap block and gapsum in the new context:

Definition 3.1 *The function gapsum(b) for a set of gaps* $b = \{g_1, .., g_k\}$ *is the sum of all integers* $g_i \in b$, *where each gap* g_i *is an integer that represents the length* $len(g_i)$ *of the*

gap:
$gapsum(b) = \sum_{i=1}^{|b|} g_i$. *We refer to the gapsum simply with* $\sum(b)$.

Definition 3.2 *A gap block* $B = \{b_i, b_j, ..b_m\}$, *where each* b_i *is a sequence of integers, each index of* $b \geq 1$ *and* $b \leq n$. $\forall b_i \forall b_j \in B : gapsum(b_i) = gapsum(b_j)$.

We now describe the algorithm to find all the blocks $P = \{B_i, .., B_k\}$ of an MSA A. A detailed example is given after the algorithms are presented.

Problem 3.2 (Finding gap blocks) *Given is a list* $L = \{l_1, .., l_n\}$ *where each* $l_i \in L$ *consists of a sequence of integers (see Figure 7). The problem is to find all possible ways to remove numbers from each* $l_i \in L$ *in the following way:*

- *chose a starting list* $l_i \in L$
- *build a gap list* b_i *by removing at most* r *numbers from list* l_i *starting from the left:* $b_i = \{l_1, .., l_k\}$ *where* $k \leq r$. $\sum(b_i)$ *is the sum of all integers in* b_i.
- $\forall$ *other lists* $l_j \in L$, *the numbers we take away have to add up the sum of the numbers we took away from list* $l_i : \sum(b_i) = \sum(b_j)$.
- *we always try to do the best: if it is possible to take away* $k \leq r$ *integers from* l_j *s.t.* $\sum(b_i) = \sum(b_j)$, *then we do it.*

In Figure 7, when r is set to 4, one possible way would be to take away the first three $\{4, 1, 4, 8\}$, which then form a block, then take away the two $\{5\}$, then remove the three $\{2, 8\}$ etc.

We now want to determine the number of possible gap blocks B that can be formed this way: we have n choices to chose the starting list $l_i \in L$. From this list l_i we can take at most r numbers. We always try to do the best to fit in as many numbers as possible for the remaining lists, so there is no choice left here. So far we have $n \cdot r$ choices.

If there are at most g numbers in each list l_i, then we can do this process at most g times (since we always take away at least one number in one list). Therefore in the worst case, the total number of possibilities is $(r * n)^g$.

The problem is exponential in numbers of gaps g, but we can restrict the size of the problem by specifying two parameters r_{max} and aa_{max} that make sense in biology:

Definition 3.3 *The parameter* aa_{max} *is the maximum number of amino acids in the MSA we look at.*

This parameter aa_{max} restricts g (the number of gaps in a block) to at most $\frac{aa_{max}}{2}$, because there cannot be more gaps than $\frac{aa_{max}}{2}$ in a stretch of aa_{max} amino acids in an MSA A. The consequence is of course that the whole algorithm has to run at most g times (we will never chose an aa_{max} that does not contain at least *one* gap).

Definition 3.4 *The parameter r_{max} is the maximum number of gaps that can be contained in any $b_i \in B$.*

So in the problem of finding gap blocks, r is restricted by r_{max}.

In real cases the number of gaps is small enough so that the problem is solvable in a reasonable amount of time for r_{max} up to 5 and aa_{max} up to about 40.

Hence the algorithms runs in $O((n \cot r_{max})^{\frac{aa_{max}}{2}} \cdot g)$ time, where g, the number of gaps in a sequence (and not n), is the crucial component for the feasibility of the algorithm.

3.1.1 Algorithms

We now describe the algorithms in detail. The first algorithm *GetBlockList* (see Algorithm 3.1.1 is the one that finds all the gap blocks. It contains the exponential part.

algorithm GetBlockList$(L, r_{max}, aa_{max}) \rightarrow P$ **is**
  $P = \{\}$
  for all $r = 1..r_{max}$ **do**
    for all $i = 1..n$ **do**
      Get r **gaps from** L_i**. This builds** b_1
      Get best block B **from** L **s.t.**
      $\forall b_i \in B : |b_i| \leq r$ **and** $gapsum(b_i) = gapsum(b_1)$
      "Now comes the exponential part $(n * r)^g$***"***
      $P = P \cup B \cup GetBlockList(L \setminus B, r_{max}, aa_{max})$
    od
  od
  return P
end

Figure 8: Algorithm 3.1

Find a list of gap blocks P from an MSA A that is already transformed to a list of integers L. P contains all possible gap blocks that are determined by the parameters r_{max} and aa_{max}

Definition 3.5 *The function $\underline{left(B)}$ is the leftmost position of any gap $g \in B$. Vice versa, the function $\underline{right(B)}$ is the rightmost position of any gap $g \in B$ plus the gap length of g.*

Definition 3.6 *A block B_i $\underline{overlaps}$ with a block B_j when $(left(B_i) - right(B_j) < 0$ and $left(B_j) - right(B_i) < 0)$ or $(right(B_i) - left(B_j) < 0$ and $right(B_j) - left(B_i) < 0)$ and if B_i and B_j share at least one sequence.*

Once we have all the gap blocks $P = \{B_1, .., B_k\}$ we can optimize the MSA A with respect to P by shifting the gaps in the blocks $B \in P$ (see Algorithms 3.1.1 and 3.1.1).

algorithm OptimizeBlock$(B, \mathrm{A}) \rightarrow \mathrm{A}_{opt}$ **is**
  for all $b_i \in B$ **do**
    $g_i' :=$ **union of all** $g \in b_i$
  od
"Each b_i ***consists of a single gap*** g_i'***."***
  $s_{max} := 0$
  $\mathrm{A}_{opt} := \mathrm{A}$
  for all $p = left(B)..right(B)$ **do**
    $\mathrm{A}' := \mathrm{A}$
    for all $g_i' \in B$ **do**
      move(g_i') **in** A' **to position** p
    od
    $s := F(\mathrm{A}')$
    if $s > s_{max}$ **then**
      $s_{max} := s$
      $\mathrm{A}_{opt} := \mathrm{A}'$
    fi
  od
  return A_{opt}
end

Figure 9: Algorithm 3.2

Shift the gaps $g \in B$ (obtained from an MSA A) such that the score $F(\mathrm{A})$ increases maximally

Definition 3.7 *The score* $\underline{S(B)}$ *of a block* $B \in \mathrm{A}$ *is the maximum possible score of the MSA* A *that can be obtained by shifting the gaps within the block by the algorithm OptimizeBlock.*

3.1.2 Example

The following example gives an intuition on how the algorithms work to find the gap blocks and on how the gaps are shifted once the gap blocks are found.

Example 3.2:[Gap blocks] In the following alignment, there are initially 14 indel events.

```
Score of the alignment: 2022.109
Maximum possible score: 2472.632

16      PDCVCPTLKQAAKAV____RLQG_QHQ____PMQ_______VRKIYQTAKHLPNVC
9       PVCVCPTLRQAARAV____SLQG_QHG____PFQ_______SRKIYKTAKYLPNIC
8       PVCVCPTLKQAARAV____SLQG_QHG____PFQ_______SRKIYQSAKYLPNIC
13      QVCVCPTLKQAAKSVRVQGQHG_________________PFQSTRIYQIAKNLPNVC
10      PLCVCPTLKGASKAVKQQVRQQLEQQG____Q____QGPHVISRIYQTATHLPKVC
7       PLCVCPTLKGASKAVKQQVRQ___QQGQQGQQLQQV_____ISRIYQTATHLPKVC
3       PLCVCPTLKGASKAVRQQVRQ___QQG___QQMQGQQMQQVISRVYQTATHLPRVC
6       PLCVCPTLKGASKAVKQQIRQQGQQQGQQGQQLQHE_____ISRIYQTATHLPRVC
4       PLCVCPTLKGAAKAVKQQIQQQGQQHGQQGQQLQHE_____IRRIYQTATHLPKVC
```

algorithm GapGrouping(A, r_{max}, aa_{max}) → A′**is**
  repeat
    $P := GetBlockList(\mathrm{A}, r_{max}, aa_{max})$
    $P := sort(P, x \rightarrow S(x))$
    "sort blocks in descending order by the score of the block"
    for all $B_i \in P$ **do**
      if $overlap(B_i, P) == false$ **then**
        A := $OptimizeBlock(B, \mathrm{A})$
      fi
    od
  od
  return A
end

Figure 10: Algorithm 3.3
Fuse and shift the gaps in each gap block B in an MSA A to get the maximum score increase for the resulting MSA.

```
12      PLCVCPTLKGASKAVKQQIQQQGQQQG__KLQM________VSRIYQTATHLPKVC
5       PLCVCPTLKGASKAVKQQIQQQGQQQG__KQQM________VSRIYQTATHLPKVC
11      PLCVCPTLRGASKAVKQQIQQQEQ__QQGKQQM________VNRIYQTATHLPKVC
```

Let r_{max} be 4 and aa_{max} be 12. With the gaps of sequences 16, 9 and 8, the following gap block can be formed:

```
16      V____RLQG_QHQ____PMQ_______V
9       V____SLQG_QHG____PFQ_______S
8       V____SLQG_QHG____PFQ_______S
```

Other values are: $\forall b \in B : gapsum(b) = 16$ and $\forall b \in B : aasum(b) = 10$.

The gaps from sequences 12, 5, and 11 can also build a gap block:

```
12      QQQG__KLQM________V
5       QQQG__KQQM________V
11      Q__QQGKQQM________V
```

After fusing the gaps (building gap blocks) according to the above rules, we end up with an MSA that has a much better score and only has 4 indel events. In addition, we can use those gaps now to get an idea of the evolutionary tree (see last section).

```
Score of the alignment: 2400.081
Maximum possible score: 2472.632

16      PDCVCPTLKQAAKAVRLQG___________QHQPMQVRKIYQTAKHLPNVC
9       PVCVCPTLRQAARAVSLQG___________QHGPFQSRKIYKTAKYLPNIC
8       PVCVCPTLKQAARAVSLQG___________QHGPFQSRKIYQSAKYLPNIC
13      QVCVCPTLKQAAKSVRVQG___________QHGPFQSTRIYQIAKNLPNVC
10      PLCVCPTLKGASKAVKQQVRQQLEQQG___QQGPHVISRIYQTATHLPKVC
7       PLCVCPTLKGASKAVKQQVRQQQGQQG___QQLQQVISRIYQTATHLPKVC
3       PLCVCPTLKGASKAVRQQVRQQQGQQM_QGQQMQQVISRVYQTATHLPRVC
6       PLCVCPTLKGASKAVKQQIRQQGQQQGQQGQQLQHEISRIYQTATHLPRVC
4       PLCVCPTLKGAAKAVKQQIQQQGQQHGQQGQQLQHEIRRIYQTATHLPKVC
```

```
12      PLCVCPTLKGASKAVKQQIQQQGQQQG_____KLQMVSRIYQTATHLPKVC
5       PLCVCPTLKGASKAVKQQIQQQGQQQG_____KQQMVSRIYQTATHLPKVC
11      PLCVCPTLRGASKAVKQQIQQQEQQQG_____KQQMVNRIYQTATHLPKVC
```

3.1.3 Island shifting

Very often algorithms produce alignments with "islands". An island is a small number of amino acids between two gaps.

Definition 3.8 *An island I is a short stretch of amino acids between two gaps. The size $len(I)$ is the number of amino acids in this island. The size is may be restricted by some parameter.*

As with the gaps, we represent each island by an integer that corresponds to the length $len(I)$ of the island. Again, the whole alignment is transformed into a list of l: $L = \{l_1, .., l_n\}$, where each l_i is a sequence of islands.

Example 3.3:

```
2       PVCVCPTLRQAAKAV____RFQGQQHQ____PEQ_______VRKIYQAAKYLPNIC
16      PDCVCPTLKQAAKAV____RLQG_QHQ____PMQ_______VRKIYQTAKHLPNVC
9       PVCVCPTLRQAARAV____SLQG_QHG____PFQ_______SRKIYKTAKYLPNIC
8       PVCVCPTLKQAARAV____SLQG_QHG____PFQ_______SRKIYQSAKYLPNIC
10      PLCVCPTLKGASKAVKQQVRQQLEQQG____Q____QGPHVISRIYQTATHLPKVC
XX      ........................other sequences ................
```

The first 4 sequence all have an island of three amino acids, and the last sequence has an island of length one. Any biologist would tell you that those islands should be pushed over to the left or right. The explanation from an evolutionary point of view is the following: it is more likely that those two events to the left and right of the island are really just one event and should therefore be merged (because they are so close to each other).

An island block D can then be defined similar to a gap block:

Definition 3.9 *$D = \{d_i, d_j, ..d_m\}$ is an island block, where each d_i is a sequence of islands, each index of $d \geq 1$ and $d \leq n$. $\forall d_i \forall d_j \in D : aasum(d_i) = aasum(d_j)$.*

The resulting algorithm is very simple: it is the same as the one for the gap blocks, except that $\forall B$ use D instead.

3.1.4 Gap stacking

Gaps appear frequently in loops as they do not have a great impact on the protein structure, as we have discussed above. To describe the algorithm for gap stacking, we need to define a distance function between two blocks B_i and B_j:

Definition 3.10 *The function* $\underline{dist(B_i, B_j)}$ *is:*
$dist(B_i, B_j) = min(|left(B_i) - \overline{right(B_j)}|, |right(B_i) - left(B_j)|)$.

In addition, we define a maximal distance $dist_{max}$ between two blocks to reduce the size of the problem. The resulting algorithm for gap stacking is very simple:

```
algorithm GapStacking(A, r_max, aa_max) → A'is
   P := GetBlockList(A, r_max, aa_max)
   s_max := 0
   for all B_i ∈ P do
      c := middle(B_i)
      for all B_j ∈ P where dist(B_i, B_j) ≤ dist_max do
         move(B_j, c)
         s := score(MSA);
         if s > s_max then
            s_max = s
            c_opt := c
         fi
      od
   od
   move blocks to optimal position c_opt
end
```

Figure 11: Algorithm 3.4
Shifts gap blocks B in an MSA A until the score $S(\mathrm{A})$ no longer increases.

4 Tree construction using gaps

Gaps can be viewed as binary state characters: each event partitions the tree into two groups, the sequences with the gap and the sequences without the gap. In addition, backmutations are extremely rare, so we can basically ignore them. This means that each event tells us which sequences belong into a subtree, and this information can be used to reconstruct the tree or at least parts of the tree. Gaps have been used before to reconstruct trees [1], but it has never been stated explicitly, so we decided to give a practical approach on how to use gaps for tree construction.
We can use the gaps to build a minimum parsimony tree [7, 8, 23, 6]. Of course we do not have any guarantee that our MSA is *correct*, so we don't know if all the gaps appear at the right spot. This leads to conflicts:

Example 4.1: The MSA in Figure 12 has four gap blocks. We use those blocks to partition the sequences (see Table 12). Event 1 partitions the tree into the sequences $\{A, B\}$ and $\{C, D, E\}$ which produces the tree in Figure 14). Events 2 and 4 do not add any information. Event 3 completes the tree (see Figure 15).

```
Event     1          2         3          4
A: RPCVCP___VLRQAAQ__QVLQRQIIQGPQQLRRLF_AA
B: RPCACP___VLRQVVQ__QALQRQIIQGPQQLRRLF_AA
C: KPCVCPRQLVLRQAAHLAQQLYQGQ____RQVRRAF_VA
D: KPCVCPRQLVLRQAAH__QQLYQGQ____RQVRRLF_AA
E: KPCLCPKQAAVKQAAH__QQLYQGQLQGPKQVRRAFRLL
```

Event	Seqs with gaps	Seqs w/o gaps
1	{A, B}	{C, D, E}
2	{A, B, D, E}	{C}
3	{C, D}	{A, B, E}
4	{A, B, C, D}	{E}

Figure 12: Partitioning of sequences using gaps

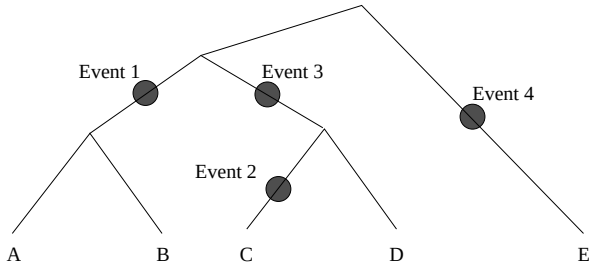

Figure 13: gap events shown in a tree

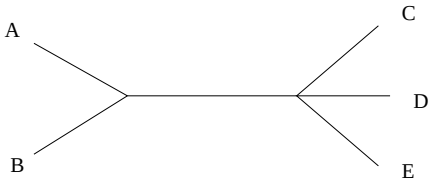

Figure 14: Tree constructed using event 1

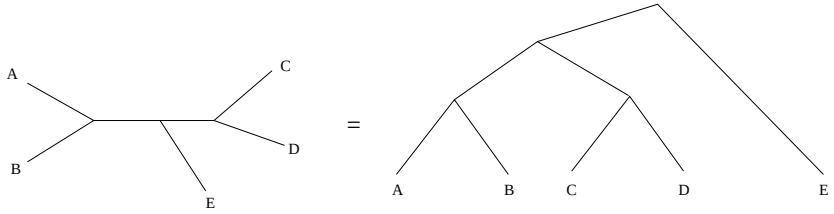

Figure 15: Tree constructed using all four events. On the left is an unrooted version, on the right a rooted version of the same tree.

To reconstruct a binary tree with n leaves completely, we need $n-3$ different splits or events, where each character state must contain at least two elements (two leaves), because a gap in just one sequences does not add any information for the tree.

4.1 Resolving conflicts

There is no guarantee that all conflicts will be resolved after gap fusion and shifting, at least if we use a heuristic. First we need to find conflicts, and as a second step we have to find a way to resolve them.

Definition 4.1 (Conflict) *Given are two events that split the leaves into the sets a_1, b_1 (event 1) and a_2, b_2 (event 2). If $a_1 \notin a_2$ and $b_1 \notin a_2$ and $a_1 \notin b_2$ and $b_1 \notin b_2$, then events 1 and 2 have a conflict.*

Example 4.2:

```
Score of the alignment: 1431.442
Maximum possible score: 1905.438

Event:       1   2    3      4      5   6   7       8    9
1       PDCVCPTLK_AAK___LQGKQQIQQQKAPQQG_QHQPMPNICQVRKIYQTAKHL
2       PVCV__TLR_AARAVKLQGKQQGQQQKAP_QQGQHLPFPNICQSRKIYKTAKYL
3       PVCV__TLK_AARIVKLQIRQQGQQQKAPQQGQHGP_FPNICQSRKIYQSAKYL
4       QVCVCPTLK_AAKAVRVQGRQQGQQQKAPQQA_QHGPFPNICQ____YQIAKNL
5       PLCVCPTLK_ASKVARQQVRQQLEQQKAP_QQGQQAPHPNICV____YQTATHL
6       PLCVCPTLKGASK___QQVR____QQKAPQQGQQLQ_QPNICVISRIYQTATHL
7       PLCVCPTLKGASK___QQVR____QQKAP_QQGQQMQQPNICVISRVY__ATHL
8       PLCVCPTLKGASK___QQIR____QQKAPQQAQQLQ_HPNICEISRIY__ATHL
9       PLCVCPTLKGAAK___QQIQQQGQQQHKAPGQQGQQQLQHPNICEIRRIYQTATHL
10      PLCVCPTLKGASK___QQIQQQGQQQKAPGQQGQQKLQPNICMVSRIYQTATHL
```

First we build a table that partitions the events from the MSA above:

Event	Seqs with gaps	Seqs without gaps	conflicts with other events
1	{2, 3}	{1, 4, 5, 6, 7, 8, 9, 10}	{5, 7}
2	{1, 2, 3, 4, 5}	{6, 7, 8, 9, 10}	{5, 6, 7}
3	{1, 6, 7, 8, 9, 10}	{2, 3, 4, 5}	{5, 6, 7}
4	{6, 7, 8}	{1, 2, 3, 4, 5, 6, 10}	{5, 7}
5	{2, 5, 7}	{1, 3, 4, 6, 8, 9, 10}	{1, 2, 3, 4 ,8 , 9}
6	{1, 4}	{2, 3, 5, 6, 7, 8, 9, 10}	{2, 3, 8}
7	{3, 6, 8}	{1, 2, 4, 5, 7, 9, 10}	{1, 2, 3, 4,9}
8	{4, 5}	{1, 2, 3, 6, 7, 8, 9, 10}	{5, 6}
9	{7, 8}	{1, 2, 3, 4, 5, 6, 9. 10}	{5, 7}

Figure 16: This table shows the conflicts that arise when gaps are used to partition the sequences in the MSA

In Table 16 we can see that events 1 and 2 do not have a conflict, because $a_1 = \{2, 3\} \in a_2 = \{1, 2, 3, 4, 5\}$, but events 1 and 5 have a conflict, because none of the sets is contained in the other.

The problem is now to resolve those conflicts. To do this we build a conflict graph: each event corresponds to a vertex, and each edge represents a conflict between two events.

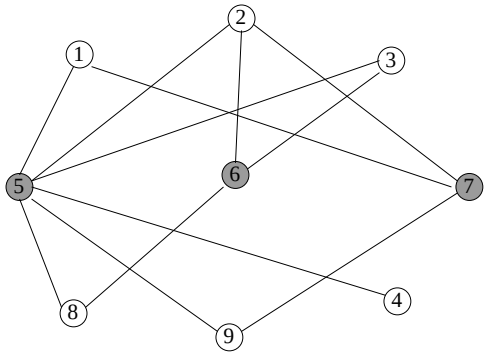

Figure 17: Gap conflict graph

Problem 4.1 (Vertex Cover) *Given is a graph $G = (V, E)$ with vertices V and edges E. The problem is to find the smallest subset $V' \in V$ such that for each edge $(a, b) \in G$: $a \in V'$ or $b \in V'$.*

We want to remove the smallest number of events to resolve the conflicts. This can be done via vertex cover. The minimal solution is {5, 6, 7}. So the conflicts are most likely caused by events {5, 6, 7}.

If we go back to the alignment we can see that the gaps of events {5, 6, 7} can be grouped into a block that partitions the tree into {9, 10} and {1, 2, 3, 4, 5, 6, 7, 8}. There are no more conflicts left, and a tree can now be constructed (see Figure 18).

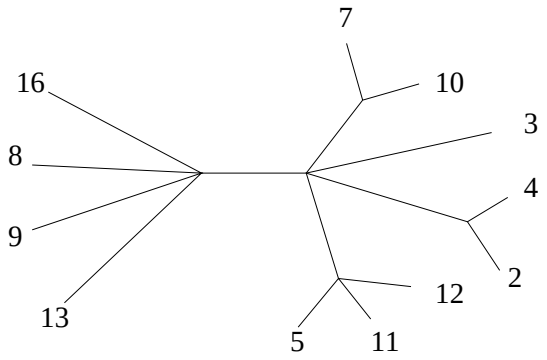

Figure 18: Tree construction of the conflict free partitions

The problem of finding the minimum vertex cover is NP-complete [20]. But usually the number of indel events is small (smaller than 20), if we restrict the size of an MSA to the size of a "domain", which is usually at most 30 to 70 amino acids long. In addition, if we restrict the number of events that could cause conflicts, there is a nice fixed parameter tractable (FPT) approach [25].

```
Score of the alignment: 1905.438
Maximum possible score: 1905.438
```

```
Event:      1    2    3       4   {5,6,7}=5'        8   9
1       PDCVCPTLK_AAK___LQGKQQIQQQKAP_QQGQHQPMPNICQVRKIYQTAKHL
2       PVCV__TLR_AARAVKLQGKQQGQQQKAP_QQGQHLPFPNICQSRKIYKTAKYL
3       PVCV__TLK_AARIVKLQIRQQGQQQKAP_QQGQHGPFPNICQSRKIYQSAKYL
4       QVCVCPTLK_AAKAVRVQGRQQGQQQKAP_QQAQHGPFPNICQ____YQIAKNL
5       PLCVCPTLK_ASKVARQQVRQQLEQQKAP_QQGQQAPHPNICV____YQTATHL
6       PLCVCPTLKGASK___QQVR____QQKAP_QQGQQLQQPNICVISRIYQTATHL
7       PLCVCPTLKGASK___QQVR____QQKAP_QQGQQMQQPNICVISRVY__ATHL
8       PLCVCPTLKGASK___QQIR____QQKAP_QQAQQLQHPNICEISRIY__ATHL
9       PLCVCPTLKGAAK___QQIQQQGQQHKAPGQQGQQLQHPNICEIRRIYQTATHL
10      PLCVCPTLKGASK___QQIQQQGQQQKAPGQQGQQKLQPNICMVSRIYQTATHL
```

Using the Traveling Salesman Ordering We can use a solution to the TSP to find a circular order in the unknown evolutionary tree $T(\mathrm{A})$, which we use in the scoring function), that corresponds to the MSA A (without knowing the tree topology) [16].

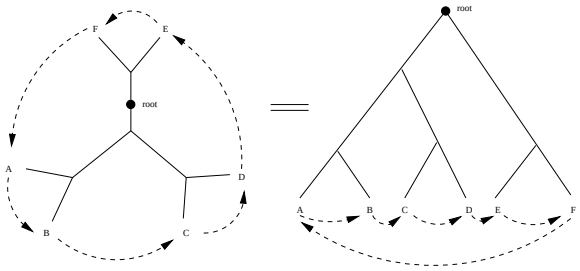

Figure 19: Circular path of a tree. On the left the tree is drawn in an unrooted version, on the right the tree is rooted.

Once we have such a TSP order C (see Definition 1.7) we can rearrange the sequences in the MSA in that order (see Figure 19). It follows that all the sequences in one subtree are grouped together, and hence all the gaps in one event (or the gaps in a block) are in one consecutive order, because an indel event always affects the whole subtree below the event.

In example 4.2, the sequences of the MSA are in circular order $(1, 2, .., 10)$. All the gaps appear as a *block*, e.g. the gaps from event 3 appear in sequences 6 through 10 and 1. Since the order is *circular*, it does not matter where we start the order.

Whenever an event causes a conflict, then the gaps of this event are usually not in this order, as can be seen in example 4.2 (the gaps of events 5, 6 and 7). The gaps are interrupted with sequences without gaps. This way we can a already identify most events that cause conflicts, which reduces the vertex cover problem. We only need to look at the gaps that do not appear in a block.

5 Discussion

In this paper we present a set of heuristic to optimize given MSAs by implicitly using structural and evolutionary information. Gaps do not appear as frequently as mutations, and back mutations are very rare. We use this fact and treat gaps as a special character. Our goal was to reduce the number of evolutionary events by fusing and shifting gaps,

hence maximizing the probability of the alignment. Once the gaps are aligned properly, a partial evolutionary tree can be reconstructed from the gaps, where conflicts can be resolved via vertex cover. The tree structure can then be resolved using other methods. All algorithms described here have been implemented into the Darwin system [13] and are available upon request. The programs can also be used via our computational biochemistry server:

`http://cbrg.inf.eth.ch`

References

[1] S. Baldauf and J. Palmer. Animals and fungi are each other's closest relatives: congruent evidence from multiple proteins. *Proc. Natl. Acad. Sci. USA*, 90(11):558 – 62, 1993.

[2] Lachlan H. Bell, John R. Coggins, and James E. Milner-White. Mix'n'match: an improved multiple sequence alignment procedure for distantly related proteins using secondary structure predictions, designed to be independent of the choice of gap penalty and scoring matrix. *Protein Engineering*, 6(7):683–690, 1993.

[3] Steven A. Benner, Mark A. Cohen, and Gaston H. Gonnet. Empirical and structural models for insertions and deletions in the divergent evolution of proteins. *J. Molecular Biology*, 229:1065–1082, 1993.

[4] Humberto Carillo and David. Lipman. The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.*, 48(5):1073–1082, 1988.

[5] Margaret O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model for evolutionary change in proteins. In Margaret O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5, pages 345–352. 1978.

[6] A. Dress and M. Steel. Convex tree realization of partitions. *Appl. Math. Lett.*, 5:3 – 6, 1993.

[7] G. Estabrook, C. Johnson, and F. McMorris. An idealized concept of the true cladistic character. *Math. Biosciences*, 23:263 – 72, 1975.

[8] G. Estabrook, C. Johnson, and F. McMorris. A mathematical foundation for the analysis of cladistic character compatibility. *Math. Biosciences*, 29:181 – 87, 1976.

[9] D. Frishman and P. Args. Incorporation of long-distance interactions into a secondary structure prediction algorithm. *Protein Engineering, in press*, 256, 1996.

[10] Dietlind L. Gerloff, Thomas F. Jenny, Lukas J. Knecht, Gaston H. Gonnet, and Steven A. Benner. The nitrogenase mofe protein, a secondary structure prediction. *FEBS*, 318(2):118–124, Mar 1993.

[11] D.L. Gerloff, C. Cohen, F. E.and Korostensky, M. Turcotte, G. Gonnet, and S.A. Benner. A predicted consensus structure for the n-terminal fragment of the heat shock protein hsp90 family. *Proteins: Struct. Funct. Genet.*, 27:450–458, 1997.

[12] Adam Godzik and Jeffrey Skolnick. Flexible algorithm for direct multiple alignment of protein structures and sequences. *CABIOS*, 10(6):587–596, 1994.

[13] Gaston H. Gonnet. A tutorial introduction to computational biochemistry using Darwin. 1994.

[14] Gaston H. Gonnet and Steven A. Benner. Probabilistic ancestral sequences and multiple alignments. In *Fifth Scandinavian Workshop on Algorithm Theory, Reykjevik July 1996*, 1996.

[15] Gaston H. Gonnet, Mark A. Cohen, and Steven A. Benner. Exhaustive matching of the entire protein sequence database. *Science*, 256:1443–1445, 1992.

[16] Gaston H. Gonnet and Chantal Korostensky. Evaluation measures of multiple sequence alignments. *J. Comp. Biol.*, 1999. submitted.

[17] O. Gotoh. An improved algorithm for matching biological sequences. *J. Mol. Biol.*, 162:705–708, 1982.

[18] Sandeep K. Gupta, John Kececioglu, and Alejandro A. Schaffer. Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. In *J. Computational Biology*, 1996.

[19] Xiaoqiu Huang. On global sequence alignment. *CABIOS*, 10(3):227–235, 1994.

[20] Richard M. Karp. *Reducibility Among Combinatorial Problems*, pages 85–103. Plenum Press, NY, 1972.

[21] B. Rost and C. Sander. Improved prediction of protein secondary structure by use of sequence profiles and neural networks. *Proc. Natl. Acad. Sci. U.S.A.*, 90:7558 – 7562, 1993.

[22] Robert B. Russell and Geoffrey J. Barton. The limits of protein secondary structure prediction accuracy from multiple sequence alignment. *J. Molecular Biology*, 234:951–957, 1993.

[23] D. Sankoff. Minimal mutation trees of sequences. *SIAM J. Appl. Math.*, 28(35 – 42), 1975.

[24] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[25] U. Stege and M. Fellows. An improved fixed-parameter tractable algorithm for vertex cover. *Technical Report, ETH Zuerich*, 318, 1999.

[26] J.D. Thompson, D.G. Higgins, and T.J Gibson. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.